מחלקת AVLTree:

הגדרנו 3 שדות למחלקה:

1. שורש – השורש של העץ
2. צומת וירטואלי – הערכים שלו זה {-1,false} הוא מתפקד בתור בנים של צמתים שאין להם בנים עם מפתחות אמיתיים ובתור האבא של השורש.
3. גודל – משתנה שסופר את כמות האיברים בעץ.

הגדרנו משתנה גלובלי לחלק מפונקציות העזר אשר יוצרות מערכים (פונקציות 10,11)

משתנה זה הוא מציין למיקום ההכנסה במערך, חתימתו indexforkeystoarray.

הסבר על הפונקציות:

פונקציות שצריך לממש:

1. empty() – מחזיר אמת או שקר אם השורש קיים או לא. סיבוכיות .
2. Search(int k) – קוראת לפונקצית find אם חזר ערך שאינו null נחזיר את שדה ה info שלו ואחרת נחזיר null. (השתמשנו בערך החזרה מסוג Boolean) סיבוכיות כסיבוכיות פונקציית העזר find שתיעדנו מטה. .
3. Insert(int k,Boolean s) – תחילה בודקים אם הקלט תקין, אם העץ ריק נקרא לinsertFirst סיבוכיות . אחרת נבצע חיפוש למפתח k. ראינו כי העלות . במידה והוא לא העץ נמשיך. נרוץ בלולאה כמו בחיפוש ונגיע למיקום שאנו רוצים להכניס את הצומת . מכניסים את הצומת ע"י עדכון פוינטרים . מתחילים את שלב התיקונים כלפי מעלה לפי האלגוריתם שלמדנו בכיתה. כמות הצמתים שנבקר חסומה ע"י גובה העץ ולכן . לבסוף כיוון שאנו יודעים שבהכנסה יש לכל היותר תיקון 1 שצריך לבצע, מבצעים תיקון לצומת שמצאנו במידת הצורך (כולל שינוי פוינטרים) . סה"כ הסיבוכיות היא עדיין .
4. Delete(int k) – תחילה נקרא לפונקצית עזר find(k) שתחזיר לנו מצביע לאיבר בעץ, סיבוכיות החיפוש . אם לא קיים איבר עם המפתח k נחזיר -1. אחרת נאתחל מונה לכמות הרוטציות. נקרא לפונקציות עזר 9 או 13 לפי המקרה שאנחנו נמצאים (תלוי האם לצומת יש 2 בנים, בן 1 או שהוא עלה). ראינו בניתוח הפונקציות כי במקרה הגרוע הסיבוכיות . לאחר המחיקה של הצומת נקרא לפונקצית עזר 4 שתעדכן את כל ה Xor של הצמתים הרלוונטיים . כעת נרוץ בלולאה מהאבא של הצומת שנמחק ועד לשורש, בכל איטרציה נרצה לדעת לאיזה מקרה באלגוריתם המחיקה שראינו הצומת שייך (האם הBF תקין והאם הגובה שלו השתנה). נקרא לבנים ונבדוק את הגובה שלהם ונעדכן את הגובה של הצומת לאחר מכן נקרא לפונקצית עזר 5 שמחזירה את ה BF ב , אם הצומת לא עבריין והגובה לא השתנה נסיים ונחזיר את המונה, אם הצומת לא עבריין והגובה כן השתנה נקרא ללולאה שוב עם האבא , אחרת נבצע רוטציה לצומת ע"י קריאה לפונקצית עזר 8 doRotation שרצה בזמן קבוע, ונחזור לתחילת הלולאה עם האבא. סה"כ אנחנו מטפסים מהצומת עד לשורש כלומר כמות האיטרציות חסומה ע"י גובה העץ ובכל איטרציה מבצעים זמן עבודה קבוע ולכן סה"כ . לאחר סיום הלולאה נחזיר את המונה. סה"כ ביצענו 4 פעולות ב והרבה פעולות עדכון שדות ושינוי פוינטרים שמתרחשות בזמן קבוע, לכן סה"כ הסיבוכיות היא .
5. Min() – מחזירים null אם העץ ריק. אחרת נמשיך מהשורש עד שנגיע לעלה כאשר בכל פעם נרד לבן השמאלי ובסוף העלה יהיה המינימום. סיבוכיות האלגוריתם חסומה ע"י גובה העץ ולכן .
6. Max() – מחזירים null אם העץ ריק. אחרת נמשיך מהשורש עד שנגיע לעלה כאשר בכל פעם נרד לבן הימני ובסוף העלה יהיה המקסימום. סיבוכיות האלגוריתם חסומה ע"י גובה העץ ולכן .
7. keysToArray() – מאתחלת מערך בגודל העץ, ובמידה שהעץ לא ריק קוראת לפונקציית עזר 10 ובכך מעדכנת את המערך שנוצר בערכים המתאימים ומחזירה אותו, סיבוכיות הפונקצייה כסיבוכיות הפונקצייה הרקורסיבית, .
8. infoToArray() – מאתחלת מערך בגודל העץ, ובמידה שהעץ לא ריק קוראת לפונקציית עזר 11 ובכך מעדכנת את המערך שנוצר בערכים המתאימים ומחזירה אותו, סיבוכיות הפונקצייה כסיבוכיות הפונקצייה הרקורסיבית, .
9. size() – מחזיר את ערך השדה size.
10. getRoot() – מחזיר את צומת השורש.
11. Successor(AVLNode node) – מחזירה את הצומת העוקב של node, במידה ויש לו בן ימני נמצא את המינימום בתת העץ הזה, נרד שמאלה עד הסוף. כלומר פעולה זו חסומה בגובה העץ. אם אין לו בן ימני, נעלה לכיוון השורש כל עוד הצומת הנוכחי הוא בן שמאלי ואמיתי (לא וירטואלי) אם מצאנו צומת מתאים נחזיר אותו אחרת נחזיר null, גם פעולה זו חסומה בגובה העץ ולכן סה"כ אפשרויות חסומות בגובה העץ, לכן .
12. prefixXor(int k) – תחילה מוצאים את הצומת עם המפתח k (מובטח לנו שהוא קיים), אנו עולים למעלה לכיוון השורש ומבצעים פעולות השוואה בוליאניות קריאה לבנים, ולמפתחות שלהם, כל אלו פונקציות ופעולות שעולות זמן קבוע ואנו מבצעים מספר חסום של פעולות אלה, בכל מעבר מעלה. סה"כ מציאת האיבר עם פונקציית עזר 1 עולה .
13. succPrefixXor(int k) – נאתחל מונה. נקרא לפונקצית עזר 5 שמחזירה את המינימום ב . כעת נרוץ בלולאה עד שנגיע לצומת שהמפתח שלו זה k כאשר בכל פעם מבצעים פעולת successor (פונקציה 11) שעולה . במקרה הגרוע k זה המפתח המקסימלי בעץ ולכן נבצע n פעולות מציאת עוקב ולכן הסיבוכיות היא .

פונקציות עזר:

1. find(int k) – בהינתן מספר k הפונקציה מחזירה את הצומת עם המפתח הזה או null אם הוא לא קיים בעץ. הפונקציה בכל שלב בודקת האם המפתח נמצא בתת עץ ימני או השמאלי וממשיכה אליו אז שהיא מגיעה לצומת המבוקש או צומת וירטואלי. האלגוריתם עובד בדיוק כפי שראינו בכיתה והוא חסום מלמעלה ע"י גובה העץ ולכן סיבוכיות היא .
2. insertFirst(int k ,Boolean i) – במידה והעץ ריק אז ניצור צומת חדש עם הערכים {k, i} ונגדיר אותו להיות השורש ונעלה את גדול העץ ב 1. סיבוכיות .
3. Update\_Xor(AVL node n)- מעדכנת את שדה ה Xor של הצומת n. .
4. Update\_Xor\_to\_root(AVL node n) – מעדכנת את ה Xor של כל הצמתים במסלול של n עד השורש. פעולת העדכון קוראת לפונקצית עזר 3 שעולה זמן קבוע, ואנו מבצעים אותה כגובה העץ, ולכן בסה"כ פעולה זו עולה .
5. getBF(AVL node n) – מחזירה את ה BF של הצומת n. .
6. rotateLeft(AVLnode n) – מקבל צומת שצריך לבצע לו רוטציה שמאלה מעדכנים את הפוינטרים לכלל הצמתים הנדרשים לפי הצורך (המקרים השונים) שראינו בכתה. פעולות עדכון פוינטרים לוקח זמן קבוע וכמות הצמתים אשר אנו משנים להם את הפוינטרים היא חסומה ואינה תלויה בגודל העץ. , בסה"כ.
7. rotateRight(AVLnode n) – מקבל צומת שצריך לבצע לו רוטציה ימינה מעדכנים את הפוינטרים לכלל הצמתים הנדרשים לפי הצורך (המקרים השונים) שראינו בכתה. פעולות עדכון פוינטרים לוקח זמן קבוע וכמות הצמתים אשר אנו משנים להם את הפוינטרים היא חסומה ואינה תלויה בגודל העץ. , בסה"כ.
8. doRotation(AVLnode n) – מקבלת צומת עבריין עם BF +-2 בודקים לאיזה מקרה מארבעת המקרים האפשריים הוא שייך (לדוגמה אם ה BF שלו +2 ושל בנו השמאלי -1 נעשה רוטציה שמאלה ואז ימינה), מבצעים את הרוטציות הנדרשות (פונקציות עזר 6 ו-7 ), שרצות בזמן קבוע ובדיקת המקרה לפי התנאים גם עולה זמן קבוע ולכן בסה"כ גם הוא רץ ב .
9. RemoveFromTreeCase1and2(AVLnode n) – פונקציית עזר לפונקצייה delete שמקבלת צומת למחיקה ובודקת- 1) אם הוא עלה 2) אם יש לו רק בן שמאלי 3) אם יש לו רק בן ימני. (case 1,2 במצגת של ההרצאה שבה האלגוריתם של המחיקה מעץ חיפוש). במידה והצומת לא עונה על אחד מן המקרים היא תחזיר false ובמידה שכן, הפונקצייה תבצע מחיקה על פי האלגוריתם שראינו בכתה למחיקת צומת מעץ, פעולות המחיקה מורכבות מקריאות לבנים, פונקציית isRealNode ופונקציות set לבנים, שהראנו שכולן רצות ב . ותחזיר true. ולכן בסה"כ תבצע .
10. Req\_keysToArray( int [] arr, AVLnode n) – פונקצייה רקורסיבית המקבלת מערך בגודל העץ ואת שורשו, ומבצעת סריקת in order בעץ שבה מכניסים את המפתחות למקומם במערך המפתחות arr לפי הסדר. אנו עוברים בסה"כ בכל קשת 2 פעמים לכל היותר ולכן בסה"כ מבצעים עבודה ליניארית בגודל העץ, .
11. Req\_InfoToArray(boolean [] arr, AVLnode n) – פונקצייה רקורסיבית המקבלת מערך בגודל העץ ואת שורשו, ומבצעת סריקת in order בעץ שבה מכניסים את ערכי הצמתים למקומם במערך הערכים arr לפי הסדר. אנו עוברים בסה"כ בכל קשת 2 פעמים לכל היותר ולכן בסה"כ מבצעים עבודה ליניארית בגודל העץ, .
12. setSize(int size) מקבלת את הערך size ומעדכנת אותו בשדה המתאים של העץ. .
13. RemoveFromTreeCase3(AVLnode n) – פונקציית עזר לפונקצייה delete שמקבלת צומת למחיקה שהוא מ case3 באופציות למחיקה (יש לו בן שמאלי ובן ימני). היא מוצאת את העוקב שלו y, מוחקת את העוקב מהעץ בעזרת פונקצית עזר 9 (בהכרח ל y אין בן שמאלי), לאחר מכן הפונקציה מכניסה את הצומת y במקום x ע"י שינוי מצביעים ומעדכנת את השורש אם נדרש. פעולות עדכון הפוינטרים לוקחות זמן קבוע, קריאת לפונקצית העזר ראינו שהיא גם לוקחת זמן קבוע. הפעולה היקרה היא מציאת העוקב שלוקחת , ולכן סיבוכיות הפונקציה היא .
14. Print -

מחלקת AVLNode:

הגדרנו 6 שדות למחלקה:

1. מפתח – מייצג את המפתח של הצומת (מספר טבעי)
2. ערך – מייצג את הערך של הצומת (true/false)
3. בן שמאלי – מייצג את הבן השמאלי של הצומת
4. בן ימני – מייצג את הבן הימני של הצומת
5. אבא – מייצג את האבא של הצומת
6. גובה – מייצג את הגובה של הצומת בעץ (מרחק מן הצומת לעלה הרחוק ביותר בתת עץ שלו).
7. Xor – מייצג את פעולת ה Xor על הצומת ושני תתי העצים שלו(משני בניו).

הסבר על הפונקציות:

1. getKey – מחזיר את המפתח של הצומת.
2. getValue – מחזיר את הערך של הצומת.
3. getLeft – מחזיר את הבן השמאלי.
4. setLeft(node n) – מקבל צומת ומעדכן את הבן השמאלי להיות הצומת.
5. getRight – מחזיר את הבן הימני.
6. setRight(node n) – מקבל צומת ומעדכן את הבן הימני להיות הצומת.
7. getparent – מחזיר את ההורה של הצומת.
8. Setparent(node n) – מקבל צומת ומעדכן את ההורה להיות הצומת.
9. IsRealNode – מחזיר אמת או שקר אם הצומת הוא אמיתי ולא וירטואלי.
10. getHeight – מחזיר את הגובה של הצומת.
11. Setheight(int k) – מעדכן את הגובה של הצומת להיות הערך k.
12. AVL node (int key, Boolean info) - בנאי המקבל מפתח מערך שלם, וערך מסוג בוליאני, ויוצר צומת חדש עם הערכים הללו.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| מספר סידורי | עלות prefixXor ממוצעת (כל הקריאות) | עלות succPrefixXor ממוצעת (כל הקריאות) | עלות prefixXor ממוצעת (100 קריאות ראשונות) | עלות succPrefixXor ממוצעת (100 קריאות ראשונות) |
| 1 | 679 | 5398 | 252 | 1150 |
| 2 | 163 | 3411 | 42 | 51 |
| 3 | 137 | 5282 | 28 | 26 |
| 4 | 127 | 8606 | 295 | 18 |
| 5 | 114 | 9352 | 5 | 13 |

מסקנות מהתוצאות-

ראשית, סיבוכיות הזמן של prefixXor היא וסיבוכיות הזמן של succPrefixXor

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| עלות הכנסה ממוצעת  מספר סידורי | עץ AVL  סדרה חשבונית | עץ ללא מנגנון איזון  סדרה חשבונית | עץ AVL  סדרה מאוזנת | עץ ללא מנגנון איזון  סדרה מאוזנת | עץ AVL  סדרה אקראית | עץ ללא מנגנון איזון  סדרה אקראית |
| 1 | 7816 | 18762 | 13004 | 15189 | 5053 | 4968 |
| 2 | 4484 | 10413 | 1414 | 879 | 2068 | 1150 |
| 3 | 2698 | 12395 | 1575 | 777 | 892 | 964 |
| 4 | 1423 | 17658 | 853 | 512 | 789 | 748 |
| 5 | 858 | 20645 | 1046 | 268 | 822 | 628 |

ציפיות:

1. כאשר מכניסים סדרה חשבונית נקבל בעץ ללא מנגנון איזון שרשרת ארוכה של מספרים ולכן גובה העץ יהיה , לעומת עץ AVL שישמור על הגובה שלו ולכן אצפה לראות זמנים טובים יותר בעץ AVL.
2. בהכנסה של סדרה מאוזנת נקבל עבור 2 העצים שהם מאוזנים ולכן כל הכנסה תעלה . אך בBST אנחנו מבצעים פחות בדיקות בפונקצית ההכנסה (לא מעדכנים גובה, לא בודקים עבריינים וכו') ולכן הקבועים שלו הם יותר טובים מעץ AVL ולכן אצפה לזמנים יותר טובים שלו.
3. בהכנסה אקראית של מספרים ראינו כי גובה העץ בממוצע יהיה עדיין , כלומר יהיה במונחים אסימפטומטיים זהה לעץ AVL אך עדיין הגובה שלו יהיה פחות טוב, מצד שני לכל הכנסה בעץ AVL כפי שהסברתי כבר יש יותר פעולות שצריך לעשות ולכן אצפה לתוצאות דומות.

מסקנות:

1. אכן כפי שציפינו זמני ההכנסה הממוצעים של עץ AVL עדיפים.
2. אכן כפי שציפינו זמני ההכנסה הממוצעים של עץ BST רגיל עדיפים.
3. אכן כפי שציפינו זמני ההכנסה הממוצעים דומים